LU08b - Python-Module für Bash

Python stellt verschiedene Module und Methoden bereit, um Bash-Befehle auszuführen und deren Ergebnisse zu verarbeiten. Die gebräuchlichsten sind:

subprocess

Das `subprocess`-Modul ist die empfohlene Methode, um externe Prozesse auszuführen. Es bietet umfangreiche Kontrolle über die Ausführung von Befehlen und die Verarbeitung ihrer Ausgaben.

Beispiel

```python
import subprocess

bash_command = 'cat /var/log/error.log | grep SEVERE'
# Den Bash-Befehl ausführen
try:
    result = subprocess.run(
        bash_command,
        shell=True,  # Use a shell to interpret the command
        check=True,  # Raise an error if the command fails
        text=True,  # Capture output as a string
        stdout=subprocess.PIPE,  # Redirect standard output
        stderr=subprocess.PIPE  # Redirect standard error
    )
    return result
except subprocess.CalledProcessError as e:
    print(f'Error executing the command: {e.cmd}')
    print(f'Error message: {e.stderr}')
    sys.exit(1)
except Exception as ex:
    print(f'An unexpected error occurred: {ex}')
    sys.exit(1)

# Ausgabe anzeigen
print(result.stdout)
```

In diesem Beispiel führt Python den Befehl `ls -l` aus und gibt das Ergebnis zurück.

os-Modul

Das `os`-Modul ermöglicht die Interaktion mit dem Betriebssystem. Für die Ausführung von Shell-Befehlen gibt es die Funktion `os.system()`. Sie ist jedoch weniger flexibel und wird oft zugunsten von `subprocess` vermieden.

Beispiel

```
import os
import sys

def run_bash_command(command):
    """
    Führt einen Bash-Befehl aus und behandelt Fehler.

    :param command: Der auszuführende Bash-Befehl als String
    """
    try:
        print(f'Running command: {command}')
        # os.system gibt den Exit-Status des Befehls zurück
        exit_status = os.system(command)

        if exit_status != 0:
            print(f'Fehler: Der Befehl '{command}' schlug fehl mit Exit-
Status {exit_status}')
            sys.exit(exit_status)
        else:
            print('Befehl erfolgreich ausgeführt!')

    except Exception as e:
        print(f'Ein unerwarteter Fehler ist aufgetreten: {e}')
        sys.exit(1)

bash_command = 'ls -l /nonexistent_directory'
run_bash_command(bash_command)
```

Beachten Sie, dass `os.system()` keine Rückgabe des Befehls erlaubt.

sh-Modul (externe Bibliothek)

Für Benutzer, die eine einfachere Syntax bevorzugen, bietet das externe `sh`-Modul eine intuitive Möglichkeit, Bash-Befehle wie Funktionen auszuführen. Es muss jedoch separat installiert werden (z. B. mit `pip install sh`).

Beispiel

```
import sh
import sys

def run_bash_command(command):
    """
    Führt einen Bash-Befehl aus und behandelt Fehler.
```

```
    :param command: Der auszuführende Bash-Befehl als String
    """
    try:
        print(f"Running command: {command}")
        # sh führt den Befehl aus und fängt automatisch Fehler ab
        result = sh.Command(command.split()[0])(*command.split()[1:])
        print(result)

    except sh.ErrorReturnCode as e:
        print(f'Fehler: Der Befehl '{command}' schlug fehl mit Fehler:
{e.stderr.decode().strip()}')
        sys.exit(e.exit_code)

    except Exception as e:
        print(f'Ein unerwarteter Fehler ist aufgetreten: {e}')
        sys.exit(1)

# Beispielbefehl
bash_command = 'ls -l /nonexistent_directory'
run_bash_command(bash_command)
```

pexpect-Modul

Wenn Sie interaktive Bash-Sitzungen automatisieren möchten (z. B. SSH, FTP oder andere Terminalsitzungen), können Sie das `pexpect`-Modul verwenden.

Beispiel

```
import pexpect

child = pexpect.spawn('ftp')
child.expect('ftp> ')
child.sendline('quit')
```

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m122/learningunits/lu08/module

Last update: **2024/12/16 08:51**